1. Introduction to edge detection and the Canny algorithm.

Edge detection is a method that locates significant changes in intensity or color within a given image. Object boundaries and significant object details typically correspond to these stark changes. Edge detection involves significant steps, which include smoothing (noise reduction), gradient computation (to find intensity changes), non-maximum suppression (to thin out detected edges), and hysteresis thresholding (to finalize and connect edges). There are many algorithms that are designed for edge detection with their own strengths and weaknesses, Canny Edge Detection being one of them.

Canny Edge Detection is a multi-step edge detection algorithm depended on for its robustness and accuracy. The steps within the algorithm are as follows:

1. **Greyscaling**: the input image is transformed by removing all color into a black and white image using the following equation: **I = 0.299 * R + 0.587 * G + 0.114 * B**
2. **Smoothing**: the grayscale image is convolved with a Gaussian filter to reduce noise and eliminate small, irrelevant details that do not need to be captured by edge detection
3. **Gradient Calculation**: using the Sobel kernel in both horizontal and vertical directions. This step highlights regions with rapid intensity changes
4. **Non-Maximum Suppression**: The gradient magnitude is scanned to find local maxima. Only the local maxima are considered as potential edges, discarding other values that do not constitute the sharpest charge in intensity
5. **Double Thresholding**: Pixels are categorized as strong, weak, or non-relevant based on their intensity values. A high threshold and a low threshold are used to classify the edges. Pixels with intensity gradient values above the high threshold are considered strong edges, while those between the low and high threshold are considered weak edges. Pixels below the low threshold are discarded.
6. **Hysteresis:** Weak edges that are connected to strong edges are considered part of the edge - helps track edges through the image.

2. Brief description of each implemented step.

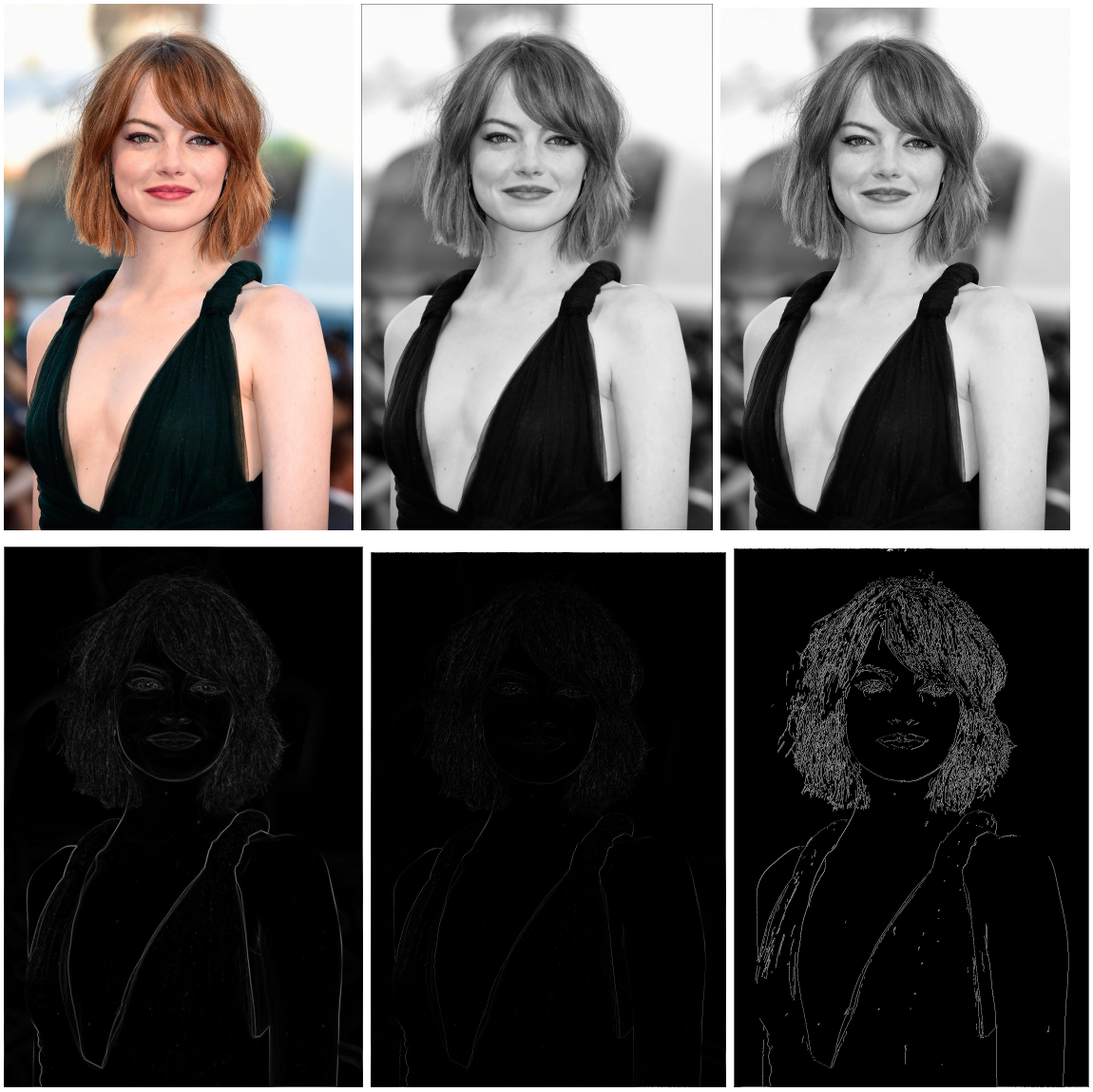
Grayscaling: Simply used np.dot to multiply the RGB value of each pixel using this formula: I = 0.299 * R + 0.587 * G + 0.114 * B

Smoothing: First defined the kernel and divided it by the sum of the elements in the matrix. Then used a nested for loop to replace each pixel by the weighted average of the 3 by 3 matrix of pixels around it by the kernel.
Gradient Calculation: FIrst defined both kernels. Then used a nested for loop to get the Gx and Gy for each pixel in the image. Then calculated the magnitude and direction using the given formulas.
Non-Maximum Suppression: Used a nested for loop to get each pixel, and compared each one to the pixel nearest to them in that direction. We found the correct pixel to compare to by brute forcing if statements. Finally, we chose the pixel that had higher intensity.

Double Thresholding: We picked a threshold for “weak” pixels, and a threshold for “strong” pixels. Used np.where to do this. Returned weak and strong pixels.

Hysteresis: Used a nested for loop to iterate through each pixel and check its strength. If weak, then we check if any neighboring pixels are strong, then we set the weak pixels into strong ones.

3. Evaluation with visual examples (e.g., plots, images).



4. Any challenges faced and how you overcame them.
   One challenge that we faced was figuring out how to apply the formulas for Gaussian filtering and gradient calculation to the pixels in the image. When doing research into how we could potentially implement them, people would often use packages that could perform the convolution for them (e.g. SciPy). After doing more research into how the algorithms work and looking at some visual examples, we were able to convolve the image using nested for loops.